Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 00:45:28 GMT
View Forum Message <> Reply to Message

David Fanning writes:

> I've proved that it is not the save/restore cycle that is
> doing this, because if the study contains just non-objects,
> say images, then there is no memory leakage. Only when the
> study contains objects do I leak.

Oh, my goodness, I love programming! You don't even have
to solve the problems. They solve themselves if you can
just bring yourself to face abject humiliation and
write about them.

After writing my last missive, I got a cup of tea
(a technique I learned from an Englishman) and sat
down to try again. The problem had disappeared!!

Sigh...Done correctly, IDL programming really is a
spiritual practice.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by Rick Towler on Wed, 03 Mar 2004 01:36:50 GMT
View Forum Message <> Reply to Message

"David Fanning" wrote ...
> David Fanning writes:
>
>> I've proved that it is not the save/restore cycle that is
>> doing this, because if the study contains just non-objects,
>> say images, then there is no memory leakage. Only when the
>> study contains objects do I leak.
>
> Oh, my goodness, I love programming! You don't even have
> to solve the problems. They solve themselves if you can
> just bring yourself to face abject humiliation and
> write about them.

>
> After writing my last missive, I got a cup of tea
> (a technique I learned from an Englishman) and sat
> down to try again. The problem had disappeared!!

Could you maybe speculate as to what was/is happening?

I was writing a response to the effect that I have seen this before too when saving and instance of IDLgrModel which contains IDLgrGraphic atoms. When I destroyed the model, (some of? all of? can't remember) the atoms were left strewn about.

My "fix" was to extract the atoms and then save them. When restoring I add them to a fresh instance of IDLgrModel.

I am curious as to why this is...

-Rick

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 02:52:40 GMT
View Forum Message <> Reply to Message

Rick Towler writes:

> Could you maybe speculate as to what was/is happening?
>
> I was writing a response to the effect that I have seen this before too when
> saving and instance of IDLgrModel which contains IDLgrGraphic atoms. When I
> destroyed the model, (some of? all of? can't remember) the atoms were left
> strewn about.
>
> My "fix" was to extract the atoms and then save them. When restoring I add
> them to a fresh instance of IDLgrModel.
>
> I am curious as to why this is...

Well, this gets curiouser and curiouser. I guess the problem is back. I don't know why I thought it had gone away. Wishful thinking, probably. Or maybe a time warp of some kind. Anyway... it's back.

Now here is the thing. This is a large application. *Everything* is an object including the "application" object.

So, the object that I am saving is a small object. It

contains three other container objects inside it,
and one of those containers contains three image objects.
So, all together, maybe a dozen objects and pointers.

Now, recall that I save it like this from within a SaveSession
method:

```
theStudy = self.currentStudy
Save, theStudy, Filename='somename.sav'
```

I restore the object like this in a RestoreSession method.

```
Obj_Destroy, self.currentStudy
Restore, Filename='somename.sav'
self.currentStudy = theStudy
```

I can get to the same place in the program either by running
a new study, in which I read some data files, etc. Or by
restoring a study. The same objects are created. The interface
looks identical.

I can destroy the main object at this point. If I read the
data to get here, I am completely clean. If I restored the
small study object to get here, I have--are you ready for this--
908 pointers and 784 objects left on the heap!!!!!

If I look at these objects and pointers I notice that my
entire application is left on the heap, even though I have
just destroyed it! In fact, the very first object on the
heap is my main program object. How can this be?

Here is a clue that may be too gruesome to contemplatee.
Every object in this system is a subclass of an IDLgrComponent
object. Lord help us, if *that* turns out to have anything to
do with this!

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 03:23:15 GMT

David Fanning writes:

> Well, this gets curiouser and curiouser.

OK, another little data point. Just before I restore
the save file where I put my simple object, I check
to see how many objects I have:

```
objects = Obj_Valid()
```

I have 306 of them. That sounds right. (Big application.)
Now I restore my simple object and check again:

```
Restore, Filename='somename.sav'
nowObjects = Obj_Valid()
```

I have 1509 of them! Whoa! What in the world is going on
here!?

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 04:50:02 GMT

David Fanning writes:

> OK, another little data point. Just before I restore
> the save file where I put my simple object, I check
> to see how many objects I have:
>
> objects = Obj_Valid()
>
> I have 306 of them. That sounds right. (Big application.)
> Now I restore my simple object and check again:
>
> Restore, Filename='somename.sav'

> nowObjects = Obj_Valid()
>
> I have 1509 of them! Whoa! What in the world is going on
> here!?

OK, are you ready for this? Are you sitting down?
What you have heard about how easy it is to save
and restore objects may be about to change.

My application has one large data set, which is stored
in an ImageCube object. The data is about 3Mbyte in size.
The StudyObject, which I have been trying all day to save,
contains a reference to the ImageCube data, but not the
data itself. By itself, it is small, maybe 100K or so.

OK, so I save the StudyObject. It obviously has a reference
to the ImageCube object, so IDL (helpfully, I think) saves
the ImageCube object, too. But my application is an object
hierarchy. Everything is more or less connected to everything
else. It's a family tree, for God's sake. We are *all* related.

IDL realizes this, thinks it is being helpful, and saves every
object in sight! And although I can't prove it, I think it
saves two backup copies as well, because the entire save file
tops out at a hefty 10 MBytes.

OK, so then I go to restore the small study object. IDL
restores *all* the objects it saved (even though most of
them are truly orphan object (widgets and what-not that
I used last Friday and which are of absolutely no use
to me now). And it also restores its two backup copies, because
no one can accuse IDL of not being thorough!

So I have 310 objects before the restore, and 1513 after the
restore. There is a difference of 1203 objects. But I can
recover these!

If I use the RESTORED_OBJECTS keyword on the RESTORE command,
I can get the references to the 1203 "extra" objects. (I am
adrift in a *sea* of objects!)

What should I do with them? Well, I *could* paw through them,
discarding the ones that are obviously bogus, but my client got
coffee while I was reading the data, so taking another coffee
break now is going to be hard on the bladder. So, I take a tip
from the Mac folks and put the darn things into a trash container.
(Really just an IDL_CONTAINER object labelled "trash".) Now, if I
take the trash out when I exit, all is well with the world and

no leaking memory.

Here is the code:

```
   ; Restore the file.
   Print, 'Objects Before: ', Obj_Valid()
   Restore, File=filename, /Relaxed_Structure_Assignment, $
     Restored_objects=helperObjects
   Print, 'Objects After: ', Obj_Valid()
   Print, "Helper Objects:', N_Elements(helperObjects)
   self.trash -> Add, helperObjects
   self.currentStudy = currentStudy
```

Cheers,

David

P.S. Let's just say tomorrow morning, I'm going back to structures. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by Craig Markwardt on Wed, 03 Mar 2004 08:22:37 GMT

View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> writes:

> IDL realizes this, thinks it is being helpful, and saves every
> object in sight! And although I can't prove it, I think it
> saves two backup copies as well, because the entire save file
> tops out at a hefty 10 MBytes.

Craigbot says: I think IDL has a cycle counting bug. If your objects are doubly- (or morely-) linked, then I'm guessing that IDL is trying to resolve the cycles, but fails. I bet if you try a simpler data structure, one without cycles, it will save fine.

But robots don't guess or bet, so I must have a logic error.

Craig-bot

--

Craig Bot Markwardt, Ph.D. EMAIL: craigmnet@REMOVEcow.physics.wisc.edu

Subject: Re: Object Madness or Restoring Nightmares
Posted by mmiller3 on Wed, 03 Mar 2004 16:09:45 GMT
View Forum Message <> Reply to Message

>>>> > "David" == David Fanning <david@dfanning.com> writes:

> Folks, Don't you just hate it when you think you understand
> something, only to realize (usually at a critical time)
> that you don't?

> Maybe I've been doing too much programming and not playing
> enough tennis lately, but I feel t-i-r-e-d. Good thing the
> ol' physical is tomorrow. Maybe I ask for some of those
> tiny blue programming pills. :-)

Take the red pill, David.

Subject: Re: Object Madness or Restoring Nightmares
Posted by Pavel Romashkin on Wed, 03 Mar 2004 16:40:27 GMT
View Forum Message <> Reply to Message

Me echoes Craig. I never had this sort of thing happen with simple
linear hierarchies. Then again, my programming style is so clean and
flawless (as opposed to David's convoluted self-linked mess), what else
do you expect :-)
Regarding Mac trash objects: check out this URL:
http://www.bushin30seconds.org/view/10_large.shtml
Uh, the ever so useful Trash.
Pavel

David Fanning wrote:
>
> David Fanning writes:
>
>> OK, another little data point. Just before I restore
>> the save file where I put my simple object, I check
>> to see how many objects I have:
>>
>> objects = Obj_Valid()
>>
>> I have 306 of them. That sounds right. (Big application.)

>> Now I restore my simple object and check again:
>>
>> Restore, Filename='somename.sav'
>> nowObjects = Obj_Valid()
>>
>> I have 1509 of them! Whoa! What in the world is going on
>> here!?
>
> OK, are you ready for this? Are you sitting down?
> What you have heard about how easy it is to save
> and restore objects may be about to change.
>
> My application has one large data set, which is stored
> in an ImageCube object. The data is about 3Mbyte in size.
> The StudyObject, which I have been trying all day to save,
> contains a reference to the ImageCube data, but not the
> data itself. By itself, it is small, maybe 100K or so.
>
> OK, so I save the StudyObject. It obviously has a reference
> to the ImageCube object, so IDL (helpfully, I think) saves
> the ImageCube object, too. But my application is an object
> hierarchy. Everything is more or less connected to everything
> else. It's a family tree, for God's sake. We are *all* related.
>
> IDL realizes this, thinks it is being helpful, and saves every
> object in sight! And although I can't prove it, I think it
> saves two backup copies as well, because the entire save file
> tops out at a hefty 10 MBytes.
>
> OK, so then I go to restore the small study object. IDL
> restores *all* the objects it saved (even though most of
> them are truly orphan object (widgets and what-not that
> I used last Friday and which are of absolutely no use
> to me now). And it also restores its two backup copies, because
> no one can accuse IDL of not being thorough!
>
> So I have 310 objects before the restore, and 1513 after the
> restore. There is a difference of 1203 objects. But I can
> recover these!
>
> If I use the RESTORED_OBJECTS keyword on the RESTORE command,
> I can get the references to the 1203 "extra" objects. (I am
> adrift in a *sea* of objects!)
>
> What should I do with them? Well, I *could* paw through them,
> discarding the ones that are obviously bogus, but my client got
> coffee while I was reading the data, so taking another coffee
> break now is going to be hard on the bladder. So, I take a tip

> from the Mac folks and put the darn things into a trash container.
> (Really just an IDL_CONTAINER object labelled "trash".) Now, if I
> take the trash out when I exit, all is well with the world and
> no leaking memory.
>
> Here is the code:
>
>   ; Restore the file.
>   Print, 'Objects Before: ', Obj_Valid()
>   Restore, File=filename, /Relaxed_Structure_Assignment, $
>   Restored_objects=helperObjects
>   Print, 'Objects After: ', Obj_Valid()
>   Print, "Helper Objects:', N_Elements(helperObjects)
>   self.trash -> Add, helperObjects
>   self.currentStudy = currentStudy
>
> Cheers,
>
> David
>
> P.S. Let's just say tomorrow morning, I'm going back to structures. :-)
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting
> Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by JD Smith on Wed, 03 Mar 2004 17:24:10 GMT
View Forum Message <> Reply to Message

On Wed, 03 Mar 2004 02:22:37 -0600, Craig Markwardt wrote:

>
> David Fanning <david@dfanning.com> writes:
>
>> IDL realizes this, thinks it is being helpful, and saves every
>> object in sight! And although I can't prove it, I think it
>> saves two backup copies as well, because the entire save file
>> tops out at a hefty 10 MBytes.
>
> Craigbot says: I think IDL has a cycle counting bug. If your objects
> are doubly- (or morely-) linked, then I'm guessing that IDL is trying
> to resolve the cycles, but fails. I bet if you try a simpler data
> structure, one without cycles, it will save fine.
>
> But robots don't guess or bet, so I must have a logic error.

>
> Craig-bot

I think it's actually simpler than that. I suspect that if you follow the train of objects containing pointers to objects with pointers etc., you'll find that some object somewhere beneath your "theStudy" object actually has a pointer or object reference to the top-level application object in it. This is very common (just keeping track of that object to consult it later on).

If this is the case, here's what happens: IDL very dutifully follows all of these downward-linking object/pointer chains, collecting and saving everything it finds on the way. This is the correct thing to do, since, as far as it knows, to have a valid "theStudy" object on disk requires all of its various holdings. Now, if at some point down the chain, IDL runs into an object which is just a convenience reference to the top level application object, it will dutifully jump right to the top of the heap and start saving the whole thing. It does know better than to save heap variables twice and how to avoid circular references; you can test this with:

```
IDL> b=obj_new('IDL_CONTAINER')
IDL> b->Add,b
IDL> save,b,FILENAME='~/b.sav'
  < new session >
IDL> help,/HEAP &  restore,'~/b.sav',RESTORED_OBJECTS=r & help,/HEAP
Heap Variables:
   # Pointer: 0
   # Object : 0
Heap Variables:
   # Pointer: 1
   # Object : 1

<ObjHeapVar1>   STRUCT    = -> IDL_CONTAINER Array[1]
<PtrHeapVar2>   STRUCT    = -> IDL_CONTAINER_NODE Array[1]
IDL> print,r
<ObjHeapVar1(IDL_CONTAINER)>
```

but simply including a convenience copy of the top level object will end up including the whole thing.

This is a problem. It's actually a bigger problem than you think, because (see the various articles on your site describing it), any object which is saved has implicit in it its class definition, so if you accidentally save 10 extra objects of different classes along with the one you're really interested in, when you restore them, any updates to any of the class definition files (class__define.pro) will never be consulted, since IDL thinks it already knows all about them. The

much-discussed solution is to explicitly resolve the class *before* restoring the object. You can find my latest incarnation of my routine which automates this here:

http://turtle.as.arizona.edu/idl/restore_object.pro

It provides some help to make sure you get the actual object you're after, and that all "helper" classes get compiled too. Sadly, you can't just inspect the objects which pop out and resolve their methods after the fact, since any changes to the actual class structure in your class__define.pro will not be noticed. This "read-once" nature of IDL classes is perhaps the biggest failing of IDL's OOP methodology. Imagine if classes could be extended at run-time? None of these restore issues would exist, and object development would be sped up immensely.

So, how do you avoid this situation? What I do is "detach" all the irrelevant data from my object before saving it. I've talked about this before, but the basic idea is (in your terms):

```
theStudy = self.currentStudy
theStudy->Save,'somename.sav'
```

with

```
pro theStudy::Save,filename
   saved_ptr=self.BigAndUselessDataPtr ; detach
   self.BigAndUselessDataPtr=ptr_new() ; a null pointer
   save, self,FILENAME=filename
   self.BigAndUselessDataPtr=saved_ptr  ; reattach
end
```

and to restore it:

```
theStudy=restore_object(file,'theStudy')
if obj_valid(theStudy) then begin
  if NOT obj_isa(theStudy,'theStudy') then $
     message,'Error restoring Study file: '+file
  ;; The study is valid
  obj_destroy,self.currentStudy
  self.currentStudy=theStudy
endif
```

This requires, of course, that you plan ahead and group all of the data that isn't necessary to include in the save file in some conveniently detachable object or pointer (or perhaps a few of them). Aside from convenience object references, widget data is a good candidate for detachment. Detaching an object reference works just the

same, but with "obj_new()" instead of "ptr_new()".

Good luck,

JD

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 20:26:36 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> I think it's actually simpler than that. I suspect that if you follow
> the train of objects containing pointers to objects with pointers
> etc., you'll find that some object somewhere beneath your "theStudy"
> object actually has a pointer or object reference to the top-level
> application object in it.

OK, I like this theory. Here is my problem. (Those of you whose eyes are already glazing over are excused. You can read the Executive Summary to follow in a couple of days.)

My study object contains three object references: one to the imageCube object that exists at the main application level and which is stored as a field in the top-level object, and two to IDL container objects. One container is empty, so don't worry about it. The other container contains ImageSlice objects, which are abstractions simply to draw a particular slice of the imageCube object. As such, they also contain references to the original imageCube object. The only other object an ImageSlice contains is a WindowIndex number reference to a drawWidgetObject, that is a child of the "self" or main object. This may be your connection back to the main object.

OK, before saving the studyObject I null out the ImageCube reference, and I go through both containers, get all the ImageSlice objects and null out their imageCube references and their WindowIndex references. The imageSlice objects now don't point to anything, and the studyObject only has references to ImageSlice objects that should be nulled out.

So now, I get the original data from the ImageCube object everyone has been pointing to, and I save it *along* with the the studyObject in a save file. (I need the data, that is the most important part of a "study".)

Are you ready? I am chagrined to see that my save file (which you remember should have been about 3 MB and was already a bloated 10 MB) is now nearly 13 MB!. And, I *still* have 1203 unnecessary

objects stored in my save file. I take it my save file is now all the objects from before *plus* the real data.

Now, *all* objects in this program inherit from a single object class. I can understand if we had to save down to that object class. But I think IDL got down to that object class (CATATOM, by the way) and saved *all* the current objects that inherit from that class! That is the only explanation that makes sense to me.

If that is true, I begin to understand some of the complexity of the iTools system, which doesn't work with objects at all, but with "descriptions" of objects. They would absolutely have to do this or they could never save and restore any of their iTool objects. Someone there must have run into this problem.

> If this is the case, here's what happens: IDL very dutifully follows all
> of these downward-linking object/pointer chains, collecting and saving
> everything it finds on the way. This is the correct thing to do, since,
> as far as it knows, to have a valid "theStudy" object on disk requires
> all of its various holdings. Now, if at some point down the chain, IDL
> runs into an object which is just a convenience reference to the top
> level application object, it will dutifully jump right to the top of the
> heap and start saving the whole thing.

I can't think how, in its current configuration, IDL could possibly get back to the top. I've been through these objects with a fine-tooth comb. There are *no* valid object references except to containers of objects that do not have valid object references.

> This is a problem. It's actually a bigger problem than you think,
> because (see the various articles on your site describing it), any
> object which is saved has implicit in it its class definition, so if you
> accidentally save 10 extra objects of different classes along with the
> one you're really interested in, when you restore them, any updates to
> any of the class definition files (class__define.pro) will never be
> consulted, since IDL thinks it already knows all about them. The
> much-discussed solution is to explicitly resolve the class *before*
> restoring the object. You can find my latest incarnation of my routine
> which automates this here:
>
> http://turtle.as.arizona.edu/idl/restore_object.pro

Alas, that EXECUTE statement makes this virtually useless to me except as an academic exercise. :-(

I've preferred not to think about this for the moment.
I'm just assuming the client is always working with a fully

compiled project so that which definition we are using is well-defined.
That will probably bite me later, as this project just never seems
to go away.

> So, how do you avoid this situation? What I do is "detach" all the
> irrelevant data from my object before saving it. I've talked about this
> before, but the basic idea is (in your terms):
>
> theStudy = self.currentStudy
> theStudy->Save,'somename.sav'
>
> with
>
> pro theStudy::Save,filename
> saved_ptr=self.BigAndUselessDataPtr ; detach
> self.BigAndUselessDataPtr=ptr_new() ; a null pointer
> save, self,FILENAME=filename
> self.BigAndUselessDataPtr=saved_ptr ; reattach
> end
>
>
> and to restore it:
>
> theStudy=restore_object(file,'theStudy')
> if obj_valid(theStudy) then begin
> if NOT obj_isa(theStudy,'theStudy') then $
> message,'Error restoring Study file: '+file
> ;; The study is valid
> obj_destroy,self.currentStudy
> self.currentStudy=theStudy
> endif
>
> This requires, of course, that you plan ahead and group all of the data
> that isn't necessary to include in the save file in some conveniently
> detachable object or pointer (or perhaps a few of them). Aside
> from convenience object references, widget data is a good
> candidate for detachment. Detaching an object reference works just the
> same, but with "obj_new()" instead of "ptr_new()".

If I understand you correctly, this is exactly what I have
tried to do, and I find myself worse off than before. I've
appealed to the IDL newsgroup because the RSI technical support
people don't exactly like to hear from me with my "big file"
examples. :-)

Cheers,

David

P.S. Let's just say if you can't reduce the problem to a 10
line program you just don't understand it well enough to ask
questions about it. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by Mark Hadfield on Wed, 03 Mar 2004 20:29:53 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
>
> [Long and thorough explanation of object saving & restoring issues]
>
> Good luck,
>
> JD

<irony>
Aww come on, JD, you haven't left anything for anyone else to say!
</irony>

PS. My spelling checker wanted to change "JD" to "CJD". Not entirely
appropriate IMHO.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: Object Madness or Restoring Nightmares
Posted by JD Smith on Wed, 03 Mar 2004 21:54:29 GMT
View Forum Message <> Reply to Message

On Thu, 04 Mar 2004 09:29:53 +1300, Mark Hadfield wrote:

> JD Smith wrote:
>>
>> [Long and thorough explanation of object saving & restoring issues]
>>

>> Good luck,
>>
>> JD
>
> <irony>
> Aww come on, JD, you haven't left anything for anyone else to say!
> </irony>
>
> PS. My spelling checker wanted to change "JD" to "CJD". Not entirely
> appropriate IMHO.

Yes, but I didn't mention object transmogrification. Talk amongst
yourselves.

JD

Subject: Re: Object Madness or Restoring Nightmares
Posted by JD Smith on Wed, 03 Mar 2004 22:15:02 GMT
View Forum Message <> Reply to Message

On Wed, 03 Mar 2004 13:26:36 -0700, David Fanning wrote:

> JD Smith writes:
>
>> I think it's actually simpler than that. I suspect that if you follow
>> the train of objects containing pointers to objects with pointers
>> etc., you'll find that some object somewhere beneath your "theStudy"
>> object actually has a pointer or object reference to the top-level
>> application object in it.
>
> OK, I like this theory. Here is my problem. (Those of you whose
> eyes are already glazing over are excused. You can read the
> Executive Summary to follow in a couple of days.)
>
> My study object contains three object references: one to the imageCube
> object that exists at the main application level and which is stored
> as a field in the top-level object, and two to IDL container objects.
> One container is empty, so don't worry about it. The other
> container contains ImageSlice objects, which are abstractions simply
> to draw a particular slice of the imageCube object. As such, they
> also contain references to the original imageCube object. The only
> other object an ImageSlice contains is a WindowIndex number reference
> to a drawWidgetObject, that is a child of the "self" or main object.
> This may be your connection back to the main object.
>
> OK, before saving the studyObject I null out the ImageCube reference,
> and I go through both containers, get all the ImageSlice objects and

> null out their imageCube references and their WindowIndex references.
> The imageSlice objects now don't point to anything, and the studyObject
> only has references to ImageSlice objects that should be nulled out.
>
> So now, I get the original data from the ImageCube object everyone has
> been pointing to, and I save it *along* with the the studyObject in
> a save file. (I need the data, that is the most important part of a
> "study".)
>
> Are you ready? I am chagrined to see that my save file (which you
> remember should have been about 3 MB and was already a bloated
> 10 MB) is now nearly 13 MB!. And, I *still* have 1203 unnecessary
> objects stored in my save file. I take it my save file is now all
> the objects from before *plus* the real data.

Hmmm, this tells me you didn't manage to prune out the problem bit. Just as an exercise, start off with your original save method, nothing funny, and start by nulling out one object or pointer at a time anywhere in theStudy. Between each such pruning, save the object and examine its contents, file size, etc. I suspect at some point you'll find the magic ladder (think chutes and ladders) which leads all the way back to the top, and your file sizes will drop and you'll have rid yourself of the bulk of those unwanted saved objects. Once you've found the wayward object or pointer, re-attach everything, descend one level into it and repeat, until you isolate the miscreant path. Of course, maybe this is what you mean when you say you went over it with a fine-toothed comb.

> Now, *all* objects in this program inherit from a single object
> class. I can understand if we had to save down to that object
> class. But I think IDL got down to that object class (CATATOM,
> by the way) and saved *all* the current objects that inherit from
> that class! That is the only explanation that makes sense to me.

That's far fetched I think. Unless this is an out-and-out bug whereby the heap descent code gets lost, I can't imagine how it would occur.

> If that is true, I begin to understand some of the complexity of
> the iTools system, which doesn't work with objects at all, but
> with "descriptions" of objects. They would absolutely have to
> do this or they could never save and restore any of their iTool
> objects. Someone there must have run into this problem.
>
>
>> If this is the case, here's what happens: IDL very dutifully follows all
>> of these downward-linking object/pointer chains, collecting and saving
>> everything it finds on the way. This is the correct thing to do, since,
>> as far as it knows, to have a valid "theStudy" object on disk requires

>> all of its various holdings. Now, if at some point down the chain, IDL
>> runs into an object which is just a convenience reference to the top
>> level application object, it will dutifully jump right to the top of the
>> heap and start saving the whole thing.
>
> I can't think how, in its current configuration, IDL could possibly get
> back to the top. I've been through these objects with a fine-tooth comb.
> There are *no* valid object references except to containers of objects
> that do not have valid object references.

Remember, it's not just objects, but any data structure (pointer,
structure, array of the same, etc.) which can contain object
references which point back to the top. Even cached messages,
anything.

>> This is a problem. It's actually a bigger problem than you think,
>> because (see the various articles on your site describing it), any
>> object which is saved has implicit in it its class definition, so if you
>> accidentally save 10 extra objects of different classes along with the
>> one you're really interested in, when you restore them, any updates to
>> any of the class definition files (class__define.pro) will never be
>> consulted, since IDL thinks it already knows all about them. The
>> much-discussed solution is to explicitly resolve the class *before*
>> restoring the object. You can find my latest incarnation of my routine
>> which automates this here:
>>
>> http://turtle.as.arizona.edu/idl/restore_object.pro
>
> Alas, that EXECUTE statement makes this virtually useless
> to me except as an academic exercise. :-(

It's only necessary if you have more than one object of the desired
class in the file, and you only want the one which corresponds to a
given variable name (like "self"). I could also use ROUTINE_NAMES
tricks to accomplish the same without EXECUTE, but that would be
cheating, i.e. instead of:

```
tmp=execute('thisvar='+var)
```

use

```
tmp=routine_names(var,FETCH=0)
```

but you didn't hear it from me.

> I've preferred not to think about this for the moment.
> I'm just assuming the client is always working with a fully
> compiled project so that which definition we are using is well-defined.

> That will probably bite me later, as this project just never seems
> to go away.

If the code never changes, that's fine. But what happens when you upgrade the class definition to include that new whiz-bang object, and the client would like to use his old saved projects with the new version? You've killed backward compatibility.

>> So, how do you avoid this situation? What I do is "detach" all the
>> irrelevant data from my object before saving it. I've talked about this
>> before, but the basic idea is (in your terms):
>>
>> theStudy = self.currentStudy
>> theStudy->Save,'somename.sav'
>>
>> with
>>
>> pro theStudy::Save,filename
>> saved_ptr=self.BigAndUselessDataPtr ; detach
>> self.BigAndUselessDataPtr=ptr_new() ; a null pointer
>> save, self,FILENAME=filename
>> self.BigAndUselessDataPtr=saved_ptr ; reattach
>> end
>>
>>
>> and to restore it:
>>
>> theStudy=restore_object(file,'theStudy')
>> if obj_valid(theStudy) then begin
>> if NOT obj_isa(theStudy,'theStudy') then $
>> message,'Error restoring Study file: '+file
>> ;; The study is valid
>> obj_destroy,self.currentStudy
>> self.currentStudy=theStudy
>> endif
>>
>> This requires, of course, that you plan ahead and group all of the data
>> that isn't necessary to include in the save file in some conveniently
>> detachable object or pointer (or perhaps a few of them). Aside
>> from convenience object references, widget data is a good
>> candidate for detachment. Detaching an object reference works just the
>> same, but with "obj_new()" instead of "ptr_new()".
>
> If I understand you correctly, this is exactly what I have
> tried to do, and I find myself worse off than before. I've
> appealed to the IDL newsgroup because the RSI technical support
> people don't exactly like to hear from me with my "big file"
> examples. :-)

I wouldn't say you're worse off, just the same (since you added
something else to the save as well). I think you'll eventually find
it. Part of the reason I communicate among objects using messages is
to avoid problems like this: instead of having to hold a reference to
an object to query it for properties on occasion, you can just
subscribe to messages which give you the relevant properties and the
right time (i.e. property push not pull). Since the messages are
dealt with as they come, they are ephemeral, and don't stick around to
cause problems like this.

JD

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 23:11:27 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> I wouldn't say you're worse off, just the same (since you added
> something else to the save as well). I think you'll eventually find
> it.

Possibly, but since I'm making about 37 cents an hour on this project
now (and it may be closer to a nickel by the time I'm finished with
it), I really can't afford more time to look. My personal opinion
is there is a bug somewhere. But in the meantime, I've turned the
damn study into a structure, saved and restored the structure, and
all is well with the world again. Did I mention I love objects. :-(

> Part of the reason I communicate among objects using messages is
> to avoid problems like this: instead of having to hold a reference to
> an object to query it for properties on occasion, you can just
> subscribe to messages which give you the relevant properties and the
> right time (i.e. property push not pull). Since the messages are
> dealt with as they come, they are ephemeral, and don't stick around to
> cause problems like this.

Well, I use messages too, but those are even more magical than
objects and I find it *really* hard to keep track of who, when,
and where when I use them. Might as well go back to prayer for
all the difficulty I have understanding what's going on.

Cheers,

David
--

David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Wed, 03 Mar 2004 23:53:25 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> I wouldn't say you're worse off, just the same (since you added
> something else to the save as well). I think you'll eventually find
> it.

Oh, dear, you may be right. :-(

There was one little coordinate object stuck *way* down there
somewhere that probably had a connection back to the surface.
In fact, two, now that I think about it. Fishing those out
is going to be trouble.

Oh, my goodness...I can see my solution unravelling again.
I *need* objects. I'm committed to objects! This is so
unbelievably ugly. When's that flight to the arctic? I may
not come back. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Thu, 04 Mar 2004 02:44:19 GMT
View Forum Message <> Reply to Message

Michael A. Miller writes:

> Take the red pill, David.

Whoa! I don't know if you are as tired of this thread
as I am, but after that pill I just don't friggin care! :-)

Here is the latest for JD to cogitate over.

One of what I used to think of as the "advantages" of my Catalyst Library is that it is an object hierarchy. If you "draw" the top-level object, all the objects below get "drawn" automatically. This means widgets appear, images get drawn in windows, coordinate systems get set up, etc. Neat.

Similarly, if you "destroy" the top-level object, all the objects below in the hierarchy get destroyed. No memory leaks, no great effort involved. Very, very neat.

But now this "feature" of my library has become a thorn in my side. (Don't worry, I'm not going to push the Mel Gibson imagery much beyond this.) If I save just one object in my save file from this "web" of objects, *all* my objects are saved. I guess that makes sense, they are all connected. But it is enormously inconvenient for me and means I can't use one of the best features of objects: the ability to store the current state of a process or operation.

Any ideas for getting out of this mess? (I see over a year of effort and very little income beginning to consolidate as a black cloud just over my right shoulder. Oh wait. That's my wife glaring at me with what look like divorce papers in her hip pocket.)

Cheers,

David

P.S. Let's just say if you had some kind of an upper-type of pill, I would be interested.

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by David Fanning on Thu, 04 Mar 2004 05:06:13 GMT
View Forum Message <> Reply to Message

David Fanning writes:

> One of what I used to think of as the "advantages" of my
> Catalyst Library is that it is an object hierarchy. If
> you "draw" the top-level object, all the objects below
> get "drawn" automatically. This means widgets appear,
> images get drawn in windows, coordinate systems get set
> up, etc. Neat.
>
> Similarly, if you "destroy" the top-level object, all the
> objects below in the hierarchy get destroyed. No memory
> leaks, no great effort involved. Very, very neat.

Ah, here is the thing about this hierarchy that you
should know. This is an object *containment* hierarchy.
The top-level object is a container that holds all the
other objects. Every object (except the top object) is
both contained in a container and can (potentially)
contain other objects. (All objects in my system
inherit IDL_CONTAINER.)

If you pick any object whatsoever out of this web,
you can (apparently easily to judge from how fast
IDL does it) traverse the entire object hierarchy.
I can see that this is the reason IDL *must* save
everything when I save even a single object that
belongs in the hierarchy.

What I can't see at the moment is a way out of
this mess.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Object Madness or Restoring Nightmares
Posted by tam on Thu, 04 Mar 2004 14:58:52 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

> David Fanning writes:
...>

> Ah, here is the thing about this hierarchy that you
> should know. This is an object *containment* hierarchy.
> The top-level object is a container that holds all the
> other objects. Every object (except the top object) is
> both contained in a container and can (potentially)
> contain other objects. (All objects in my system
> inherit IDL_CONTAINER.)
>
> If you pick any object whatsoever out of this web,
> you can (apparently easily to judge from how fast
> IDL does it) traverse the entire object hierarchy.
> I can see that this is the reason IDL *must* save
> everything when I save even a single object that
> belongs in the hierarchy.
>
> What I can't see at the moment is a way out of
> this mess.
>
> Cheers,
>
> David

Hi David,

Forgive me if I'm asking stupid questions... (OK the if is superfluous!)

Clearly each object contains pointers to all of its children
so if you save the parent all the objects contained in it
are saved. But I don't see why a child (still taking about the containment
hierarchy, not the inheritance tree) needs to point to its
parent? Where is that pointer coming from and what is it doing?

I gather that each object needs to point to the class definition of the
top level container since that's also the class definition of
the root of the inheritance tree, but I wouldn't have thought
that saving the definition of the class means that you
have to save every instance of the class. That would certainly
seem like a broken implementation for the SAVE functionality.

Regards,
Tom McGlynn

Subject: Re: Object Madness or Restoring Nightmares
Posted by JD Smith on Thu, 04 Mar 2004 17:57:18 GMT
View Forum Message <> Reply to Message

On Wed, 03 Mar 2004 22:06:13 -0700, David Fanning wrote:

> David Fanning writes:
>
>> One of what I used to think of as the "advantages" of my
>> Catalyst Library is that it is an object hierarchy. If
>> you "draw" the top-level object, all the objects below
>> get "drawn" automatically. This means widgets appear,
>> images get drawn in windows, coordinate systems get set
>> up, etc. Neat.
>>
>> Similarly, if you "destroy" the top-level object, all the
>> objects below in the hierarchy get destroyed. No memory
>> leaks, no great effort involved. Very, very neat.
>
> Ah, here is the thing about this hierarchy that you
> should know. This is an object *containment* hierarchy.
> The top-level object is a container that holds all the
> other objects. Every object (except the top object) is
> both contained in a container and can (potentially)
> contain other objects. (All objects in my system
> inherit IDL_CONTAINER.)
>
> If you pick any object whatsoever out of this web,
> you can (apparently easily to judge from how fast
> IDL does it) traverse the entire object hierarchy.
> I can see that this is the reason IDL *must* save
> everything when I save even a single object that
> belongs in the hierarchy.
>
> What I can't see at the moment is a way out of
> this mess.

Why not implement a set of methods in your top-level which leverages the inherent connectedness to detach unnecessary objects before saving? The only technical problem is where to stick the detached objects while you save (you can't stick them somewhere else in the object: you'll be back to the same problem). This can be accomplished by dynamically building a list of objects and their detached components and propagating it all the way up to a variable at the top stack level at the time of the call. Something like this:

```
pro topClass::Detach, detachlist, RECORD=rec
  if obj_valid(self.parent) then begin
     ;; Add to or create this object's "detached" record
     if n_elements(rec) gt 0 && ptr_valid(rec.Detached) then $
        *rec.Detached=create_struct('parent',self.parent,*rec.Detach ed) $
     else rec={Object:self, Detached: ptr_new({parent: self.parent})}
```

```
    self.parent=obj_new()
  endif
  ;; Stow our detached set on the list
  if n_elements(rec) gt 0 then begin
    if n_elements(detachlist) gt 0 then detachlist=[detachlist,rec] else
    detachlist=[rec]
  endif
  if ptr_valid(self.children) then $
    for i=0,n_elements(*self.children)-1 do $
      (*self.children)[i]->Detach,detachlist
end

pro topClass::Reattach, detached
  self.parent=(*detached).parent
  ptr_free,detached
end

pro topClass::ReattachList, list
  for i=0,n_elements(list)-1 do $
    if obj_valid(list[i].Object) && ptr_valid(list[i].Detached) $
    then list[i].Object->Reattach,list[i].Detached
end

pro topClass::Save, file, COMPRESS=comp
  self->Detach,list             ;all our vital info is now stashed in list
  catch, serr
  if serr ne 0 then begin       ;it failed!
    catch,/CANCEL
    self->ReattachList,list
    message,'Error Saving to File: '+file
  endif
  save,self,FILENAME=file,COMPRESS=comp
  catch,/CANCEL
  self->ReattachList, list
end
```

Now, when you say 'obj->Save, file', it will detach its unnecessary parts, stowing them on the list for safe keeping, and then recursing down to its children and so on, thus sending a propagating wave of detachment all the way down the tree hierarchy. Then the object and its children will be saved, and then everything will be re-attached by iterating over the detached list. Notice how I was careful to reattach everything in case of error too.

Now suppose some lower class has more than just the parent that it needs to detach, e.g. some widget ids, irrelevant to keep track of, since of course they will change. Then you can simply overload the Detach and Reattach methods like so:

```
pro lowerClass::Detach,list, RECORD=rec
  if n_elements(rec) gt 0 && ptr_valid(rec.Detached) then $
     *rec.Detached=create_struct('widget_info',self.widget_info,* rec.Detached)$
  else rec={Object:self, Detached: ptr_new({widget_info: self.widget_info})}
  self.widget_info=ptr_new()
  self->topClass::Detach,list,RECORD=rec
end

pro lowerClass::Reattach, detached
  self.widget_info=(*detached).widget_info
  self->topClass::Reattach, detached
end
```

Here I allow for even further sub-classing with the same RECORD keyword.
Anyway, this (or rather some tested and debugged version of this),
should serve well enough to strip out all those troublesome back links
for saving. Of course, when you restore the object from disk, none of
the parent references will be valid, but perhaps this is not a problem,
if you're just using the data embedded in this structure. Another
option which is fancier but doable is to only trim parents which point
"above" you in the tree hierarchy. I leave that one as an exercise ;).

JD
